ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SECONDA FACOLTÀ DI INGEGNERIA
Corso di Laurea Triennale in Ingegneria Elettronica, Informatica e Telecomunicazioni

IL RUOLO DELLA PIANIFICAZIONE DEL COLLAUDO NELLO SVILUPPO DEL SOFTWARE IN AMBIENTE ANDROID

Elaborata nel corso di: Ingegneria del software

Tesi di Laurea di:
KARIN PASINI

Relatore:
Prof. ANTONIO NATALI

ANNO ACCADEMICO 2011–2012
II SESSIONE

Indice

Introduzione

In questo elaborato verrà analizzato un case study nato dall'esperienza di tirocinio, in cui si sono potute osservare le dinamiche di lavoro all'interno di un'azienda che produce software e delle maggiori problematiche che investono questo tipo di attività.
Inanzitutto si vuole chiarire cos'è il software. Esso consiste nel codice delle istruzioni e i dati necessari a realizzare delle funzioni su un computer, ma non solo: include anche tutte le rappresentazioni di queste istruzioni e dati. In particolare, le rappresentazioni non includono soltanto il codice sorgente del programma e i file dati, ma modelli creati durante le attività di analisi e progettazione [McGregor,Sykes 1]. Il piano di collaudo copre un importante ruolo nella creazione di queste rappresentazioni.
L'elaborato vuole porre i riflettori sulla pianificazione del collaudo e sull'importanza che ha nel processo di sviluppo del software. Un'attenta pianificazione del collaudo permette di formalizzare inequivocabilmente lo scopo del sistema, di precisare senza possibilità di incomprensione cosa devono fornire le funzionalità del sistema. Il linguaggio naturale, infatti, non è sufficiente a questo scopo, in quanto con il suo uso si può equivocare il significato delle parole, mentre l'obiettivo della pianificazione del collaudo è non lasciare spazio alle interpretazioni.
La pianificazione del collaudo inizia durante l'analisi del sistema, nel momento in cui devono essere definite le funzionalità del sistema. Questo incentiva il lavoro cooperativo, perchè l'analista, una volta completato il lavoro di analisi e definizione del piano di collaudo, potrebbe delegare a un progettista la realizzazione del sistema, che si baserebbe su concetti definiti formalmente. Non è però escluso che, durante le fasi di progettazione e implementazione, il progettista evidenzi aspetti o problematiche del sistema che precedentemente non erano stati presi sotto esame, e quindi modifichi il piano di collaudo. Per questo la pianificazione del collaudo non è una fase fissa e non si inserisce in un processo a cascata, come non lo è tutto il processo di produzione di un sistema software, che risulta invece un procedimento dinamico all'interno di un processo a spirale.
Mediante la pianificazione del collaudo è possibile determinare in modo formale cosa

deve fare il sistema, ovvero senza scendere nei particolari implementativi. In questo modo, il piano di collaudo astrae dalla particolare tecnologia con cui verrà realizzato il sistema, e si pone ad un livello più alto di espressività. Grazie a questo, la pianificazione può essere considerata come base su cui fondarsi per realizzare il sistema nelle più diverse tecnologie.
L'elaborato mira a evidenziare come la pianificazione del collaudo possa risultare utile per affrontare in modo sistematico l'analisi del problema, definire gli scenari applicativi e fornire un punto di riferimento per l'organizzazione del processo di produzione.
I vantaggi tratti dalla pianificazione non sono però così facili da raggiungere: essere un buon collaudatore infatti è più difficile che essere un bravo sviluppatore perchè collaudare non richiede soltanto una precisa conoscenza del processo di sviluppo e dei suoi prodotti, ma comporta anche l'abilità di anticipare possibili guasti ed errori. Un collaudatore deve approcciarsi al software con un atteggiamento che mette in discussione tutto ciò che riguarda il software [McGregor,Sykes 1].

L'elaborato di tesi si sviluppa in queste parti:

- Primo capitolo. Presentazione del caso di studio e Analisi dei requisiti.
- Secondo capitolo. Analisi del problema.
- Terzo capitolo. Piano di collaudo.
- Quarto capitolo. Progetto e implementazione.
- Conclusioni.

Capitolo 1

Presentazione del caso di studio e Analisi dei requisiti

Il caso di studio preso in esame nell'elaborato è tratto dell'esperienza di tirocinio. Un'azienda produttrice di software riceve una commissione da parte di un'azienda di macchine per ufficio riguardante la progettazione e realizzazione di un'applicazione in ambiente Android. L'azienda produttrice di software, dopo un colloquio con il titolare dell'azienda di macchine per ufficio, redige i requisiti che l'applicazione deve soddisfare.

1.1 Requisiti

Un'azienda di macchine per ufficio si occupa di vendita e assistenza tecnica ai clienti che hanno acquistato dei prodotti. La gestione dell'assistenza tecnica (inserimento interventi, evasione interventi ecc.) è demandata ad un applicativo realizzato dall'azienda, che opera accedendo e modificando un Database in cui sono registrati tutti gli interventi e i prodotti dell'azienda. L'azienda vuole realizzare un'applicazione mobile che permetta ai suoi tecnici di accedere al Database per consultare gli interventi a proprio carico tramite uno smartphone o un tablet. L'accesso al Database avverrà dopo la localizzazione spaziale del tecnico in modo da poter visualizzare gli interventi nelle vicinanze del tecnico. La piattaforma su cui verrà realizzata l'applicazione è Android, che negli ultimi tempi sta prendendo sempre più piede nelle soluzioni mobile.

L'analisi dei requisiti viene formalizzata nella definizione di un glossario insieme ai casi d'uso, descritti da scenari.

1.2 Glossario

Prodotto	Prodotto acquistato da un cliente presso l'azienda
Intervento	Richiesta di assistenza tecnica su un prodotto di un cliente
Database	Database in cui sono registrati tutti i prodotti, le chiamate e le posizioni dei clienti
Tecnico	Esegue le chiamate di assistenza, accede al Database per visionare le proprie chiamate

Tabella 1.1: Glossario

1.3 Casi d'uso

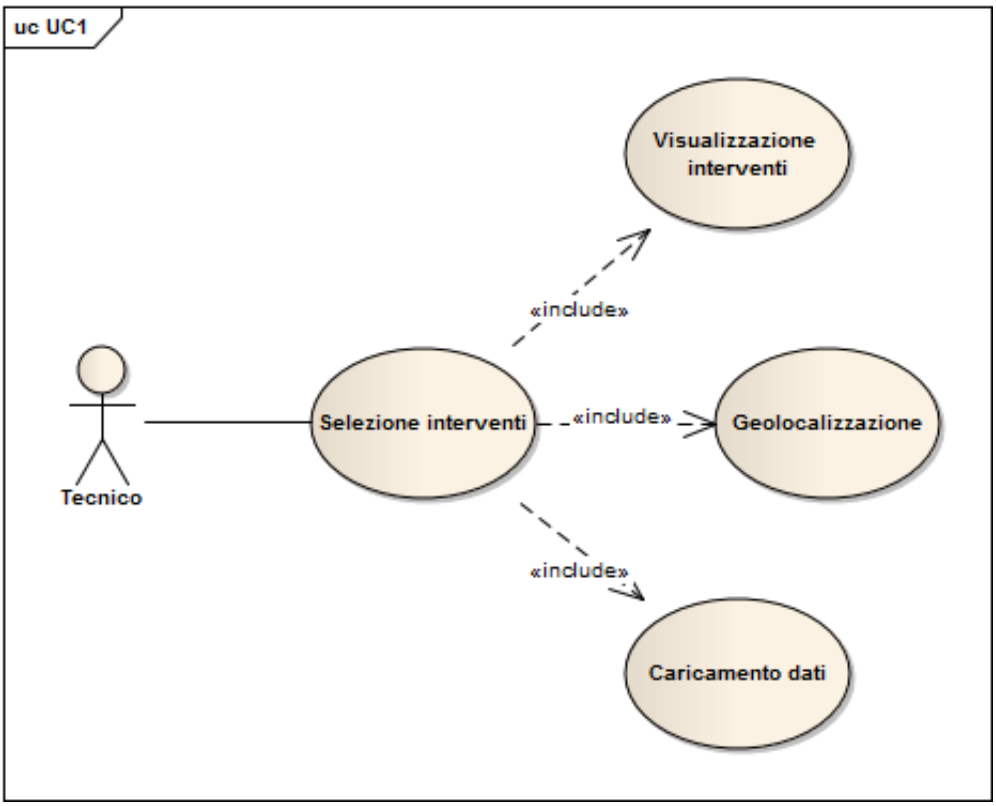

Figura 1.1: UC1

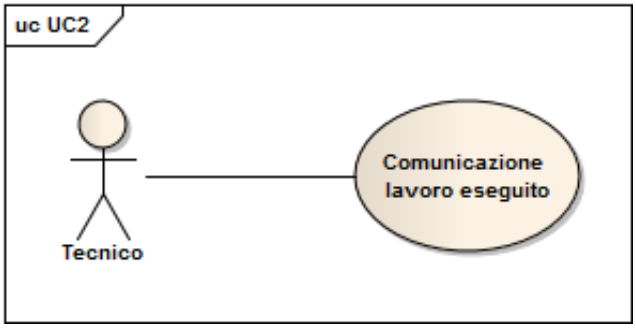

Figura 1.2: UC2

1.4 Scenari

ID	UC1
Nome	Selezione interventi
Descrizione	Il tecnico accede al Database per consultare gli interventi assegnatigli
Attori	Il tecnico
Precondizioni	Il Database deve essere stato caricato con i dati di interesse
Corso principale	Il sistema geolocalizza il tecnico e riporta la lista degli interventi assegnatigli
Corso alternativo	Il sistema non riesce a geolocalizzare il tecnico o non riesce a connettersi al Database
Condizioni di esito positivo	Il tecnico può consultare la lista degli interventi
Condizioni di esito negativo	Il tecnico non può consultare la lista degli interventi

Tabella 1.2: S1-UC1

ID	UC2
Nome	Comunicazione lavoro eseguito
Descrizione	Il tecnico comunica al Database il completamento dell'intervento
Attori	Il tecnico
Precondizioni	Il tecnico deve aver completato l'intervento
Corso principale	Il sistema modifica il Database inserendo il completamento dell'intervento
Corso alternativo	Il sistema non riesce a connettersi al Database
Condizioni di esito positivo	Il Database è stato correttamente modificato
Condizioni di esito negativo	Il Database non è stato modificato

Tabella 1.3: S2-UC2

1.5 Modello del dominio e Architettura del sistema

Il sistema può essere diviso in sottosistemi, precisamente tre che formano una three-tier architecture. Architettura three-tier ("a tre strati") indica una particolare architettura software che prevede la suddivisione del sistema in tre diversi moduli dedicati rispettivamente alla interfaccia utente, alla logica funzionale (business logic) e alla gestione dei dati persistenti. Tali moduli sono intesi interagire fra loro secondo le linee generali del paradigma client-server (l'interfaccia è cliente della business logic, e questa è cliente del modulo di gestione dei dati persistenti) e utilizzando interfacce ben definite. In questo modo, ciascuno dei tre moduli può essere modificato o sostituito indipendentemente dagli altri [3-tier].

- SMART CLIENT: Il cuore dell'applicazione, gestisce le funzionalità del sistema;
- WEB SERVER: Server che si occupa di connettersi al Database, interrogarlo e modificarlo;
- DB: Database che contiene le informazioni utili all'utente;

Il Database viene gestito da un sistema esterno (GestoreDB).

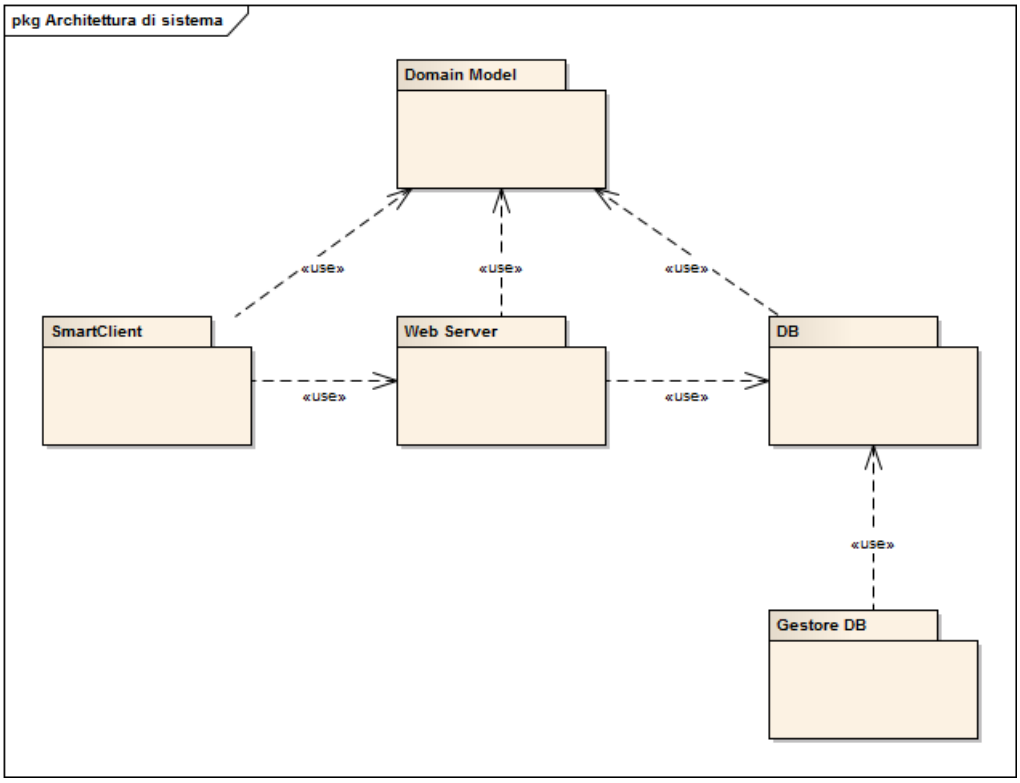

Figura 1.3: Architettura di sistema

L'UML non è il linguaggio migliore per esprimere il concetto di sottosistemi, in questo caso sono stati rappresentati tramite package. Per ora ci basta identificare i 3 sottosistemi e i ruoli che coprono.
Tutti i sottosistemi fanno uso di entità che compongono il dominio, illustrato tramite un diagramma UML.

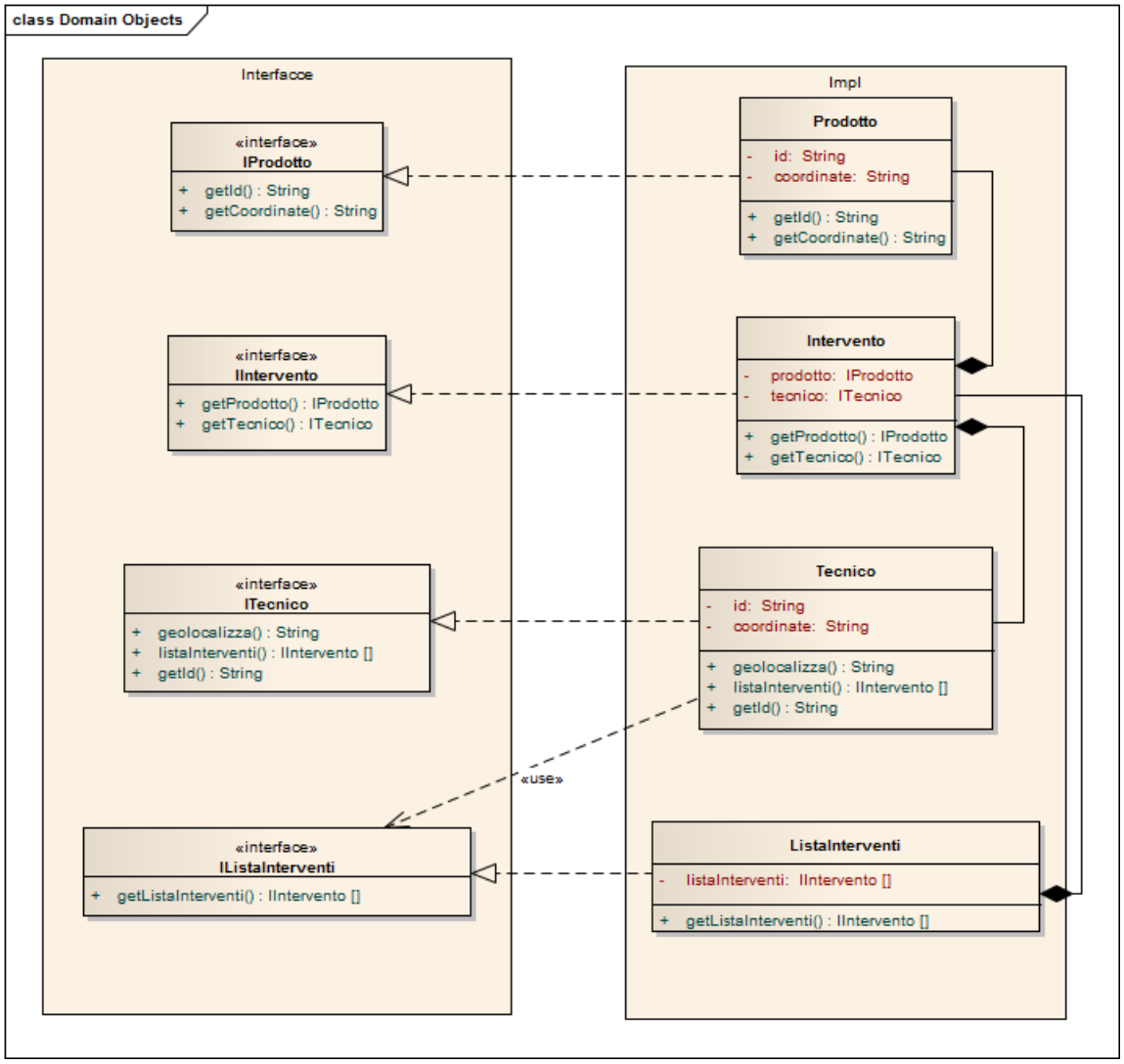

Figura 1.4: Modello del dominio

1.6 Unit Test

A questo punto si può iniziare a parlare di piano di collaudo. In particolare si può descrivere uno "Unit Test", in cui vengono definite le caratteristiche delle singole unità del dominio. Esso permette di essere sicuri del fatto che i singoli componenti che costituiscono un'applicazione funzionino correttamente se utilizzati in modo isolato, senza

quindi dipendenze dagli altri componenti con cui collaborano [Carli].
Di seguito è proposto un possibile Unit Test per il modello del dominio.

Unit Test per l'entità Prodotto.

```
public interface IProdotto {

    public String getId();
    public String getCoordinate();
}
```

Figura 1.5: Interfaccia IProdotto

```
public class Prodotto implements IProdotto{

    private String id;
    private String coordinate;

    public Prodotto(String id, String coordinate) {
        this.id=id;
        this.coordinate=coordinate;
    }

    public String getId() {
        return id;
    }

    public String getCoordinate() {
        return coordinate;
    }

}
```

Figura 1.6: Classe Prodotto

```
public class ProdottoTest extends junit.framework.TestCase{

    protected IProdotto prodotto;

    public void setUp() throws Exception {
        prodotto=new Prodotto("01","44.5,12.3");
    }

    public void testGetId() {
        assertEquals(prodotto.getId(),"01");
    }

    public void testGetCoordinate(){
        assertEquals(prodotto.getCoordinate(),"44.5,12.3");
    }

}
```

Figura 1.7: Unit Test Prodotto

Unit test per l'entità Tecnico.

```
public interface ITecnico {

    public String getId();
    public IIntervento[] listaInterventi();
    public String geolocalizza();
}
```

Figura 1.8: Interfaccia ITecnico

```java
public class Tecnico implements ITecnico{

    private String id;
    private String coordinate;

    public Tecnico(String id, String coordinate){
        this.id=id;
        this.coordinate=coordinate;

    }

    public String getId() {
        return id;
    }

    //questo metodo non può essere definito tramite lo Unit Testing
    //perchè richiede l'interazione con altri componenti
    public IIntervento[] listaInterventi() {
        return null;
    }

    public String geolocalizza() {
        return coordinate;
    }
}
```

Figura 1.9: Classe Tecnico

```java
public class TecnicoTest extends junit.framework.TestCase{

    protected ITecnico tecnico;

    public void setUp() throws Exception {
        tecnico=new Tecnico("010", "42.02,13.3");
    }

    public void testGetId(){
        assertEquals(tecnico.getId(),"010");
    }

    public void testGeolocalizza(){
        assertEquals(tecnico.geolocalizza(), "42.02,13.3");
    }
}
```

Figura 1.10: Unit Test Tecnico

Unit test per l'entità Intervento.

```
public interface IIntervento {

    public IProdotto getProdotto();
    public ITecnico getTecnico();
}
```

Figura 1.11: Interfaccia IIntervento

```
public class Intervento implements IIntervento{

    private IProdotto prodotto;
    private ITecnico tecnico;

    public Intervento(IProdotto prodotto, ITecnico tecnico){
        this.prodotto=prodotto;
        this.tecnico=tecnico;
    }

    public IProdotto getProdotto() {
        return prodotto;
    }

    public ITecnico getTecnico() {
        return tecnico;
    }

}
```

Figura 1.12: Classe Intervento

```
public class InterventoTest extends junit.framework.TestCase{

    protected IIntervento intervento;
    protected IProdotto pr;
    protected ITecnico t;

    public void setUp() throws Exception {
        pr=new Prodotto("01", "34.8,11.2");
        t=new Tecnico("020", "29.3,3.9");
        intervento=new Intervento(pr,t);
    }

    public void testGetProdotto(){
        assertEquals(intervento.getProdotto(), pr);
    }

    public void testGetTecnico(){
        assertEquals(intervento.getTecnico(), t);
    }
}
```

Figura 1.13: Unit Test Intervento

Unit test per l'entità ListaInterventi.

```
public interface IListaInterventi {

    public IIntervento[] getListaInterventi();

}
```

Figura 1.14: Interfaccia IListaInterventi

```
public class ListaInterventi implements IListaInterventi{

    private IIntervento[] listaInterventi;

    public ListaInterventi(IIntervento [] lista){
        this.listaInterventi=lista;
    }

    public IIntervento [] getListaInterventi() {
        return listaInterventi;
    }

}
```

Figura 1.15: Classe ListaInterventi

```
public class ListaInterventiTest extends junit.framework.TestCase{

    protected IListaInterventi lista;
    protected IIntervento[] i;

    public void setUp() throws Exception {
        IProdotto p1=new Prodotto("01", "34.8,11.2");
        IProdotto p2=new Prodotto("02", "64.8,31.2");
        ITecnico t1=new Tecnico("020", "29.3,3.9");
        ITecnico t2=new Tecnico("030", "26.7,13.5");
        IIntervento i1=new Intervento(p1,t1);
        IIntervento i2=new Intervento(p2,t2);
        i =new IIntervento[2];
        i[0]=i1; i[1]=i2;
        lista=new ListaInterventi(i);
    }

    public void testGetListaInterventi(){
        assertEquals(lista.getListaInterventi(),i);
    }

}
```

Figura 1.16: Unit Test ListaInterventi

Capitolo 2

Analisi del problema

I sottosistemi identificati interagiscono tra di loro in diverse modalità, come rappresentato nel diagramma di sequenza.

- SMART CLIENT-WEB SERVER: Lo Smart Client invia una richiesta al Web Server tramite protocollo http. Quando il Web Server ha ottenuto la lista degli interventi la riporta allo Smart Client, sempre tramite protocollo http.
- WEB SERVER-DB: Il Web Server si connette al DataBase ed esegue una query su di esso. Il DataBase risponderà con il risultato della query.

N.B. Tutte queste interazioni sono effettuate in remoto, pertanto è necessaria una connessione Internet.

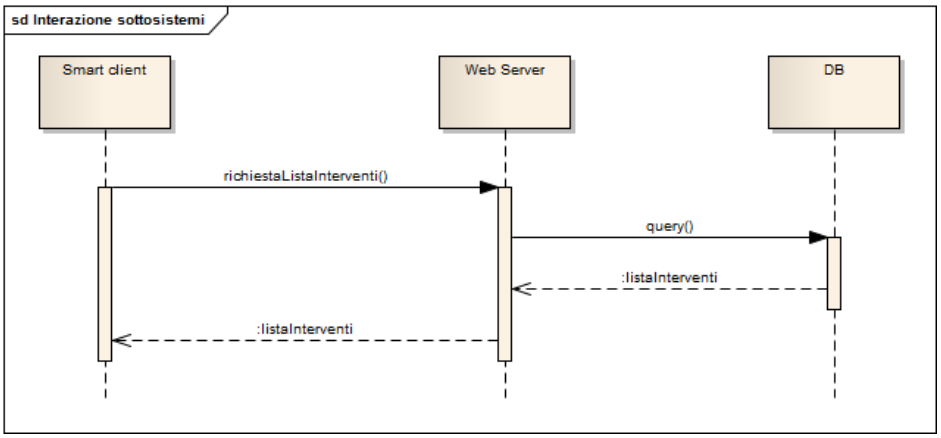

Figura 2.1: Interazione sottosistemi

Nell'analisi dei requisiti sono emerse diverse problematiche riguardanti il buon esito della consultazione della lista degli interventi da parte del tecnico. Esse riguardavano:

Geolocalizzazione. Se il sistema non riesce a geolocalizzarsi, la lista degli interventi non può essere fornita.

Connessione. Se non c'è connessione, il sistema non riesce a contattare il Web Server e di conseguenza non può connettersi al Database. Inoltre, può essere dispendioso e poco efficiente utilizzare il traffico dati tutte le volte che si richiede la lista degli interventi.

Per quanto riguarda la prima problematica, non emergono possibili strumenti risolutivi per questa eventualità.
Il problema della connessione invece, può essere ovviato rendendo disponibile un sottoinsieme di dati del Database in locale, in modo da poter usufruirne anche in assenza di rete e senza costi. Ovviamente i dati in locale in certi momenti possono non essere allineati con il Database remoto, a causa di aggiornamenti del Database, ma è ritenuto comunque un buon compromesso. A questo punto quindi si presentano due modalità diverse di accesso ai dati: una modalità in remoto e una in locale, e bisogna stabilire in quali momenti utilizzare l'una o l'altra. Non è possibile demandare all'applicazione il compito di decidere quale modalità attuare, perchè non sarebbe in grado di gestire gli eventuali criteri che porterebbero a una scelta rispetto che all'altra, per cui è ritenuto opportuno lasciare all'utente la possibilità di adottare la modalità più adatta in quel momento, tramite una specifica interfaccia.

2.1 Il sottosistema SmartClient

Il sottosistema SmartClient rappresenta la parte di sistema di maggior interesse. Esso è costituito da tre parti, illustrate nell'immagine sottostante.

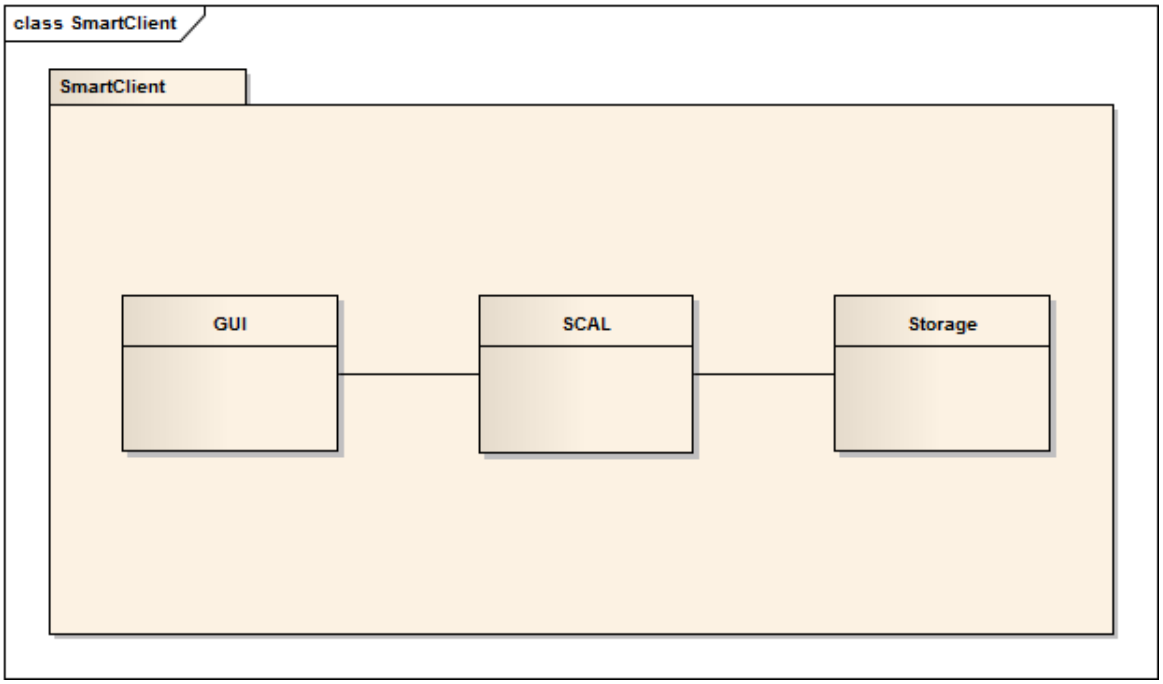

Figura 2.2: Il sottosistema SmartClient

- GUI: l'utente utilizza questa interfaccia per accedere alle funzionalità del sistema;
- SmartClient Application Logic (SCAL): realizza le funzionalità del sistema e interagisce con il Web Server;
- Storage: repository locale di dati. Dato che il sistema verrà realizzato in ambiente Android, si potrebbe pensare ad un Content Provider che fornisca le funzionalità dello Storage. In questo modo si risolverebbe il problema della connessione citato precedentemente.

Una volta definita la struttura dello SmartClient, si passa alla descrizione del suo funzionamento, tramite diagrammi di sequenza.

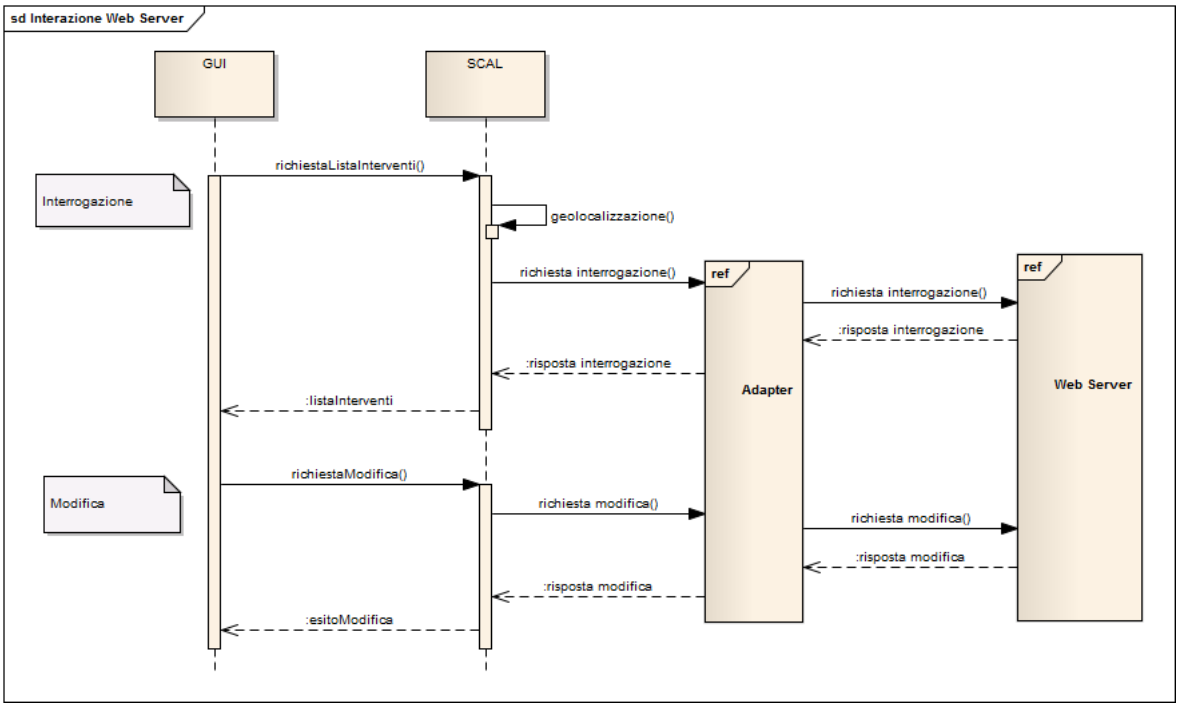

Figura 2.3: Struttura dell'interazione SmartClient-Web Server

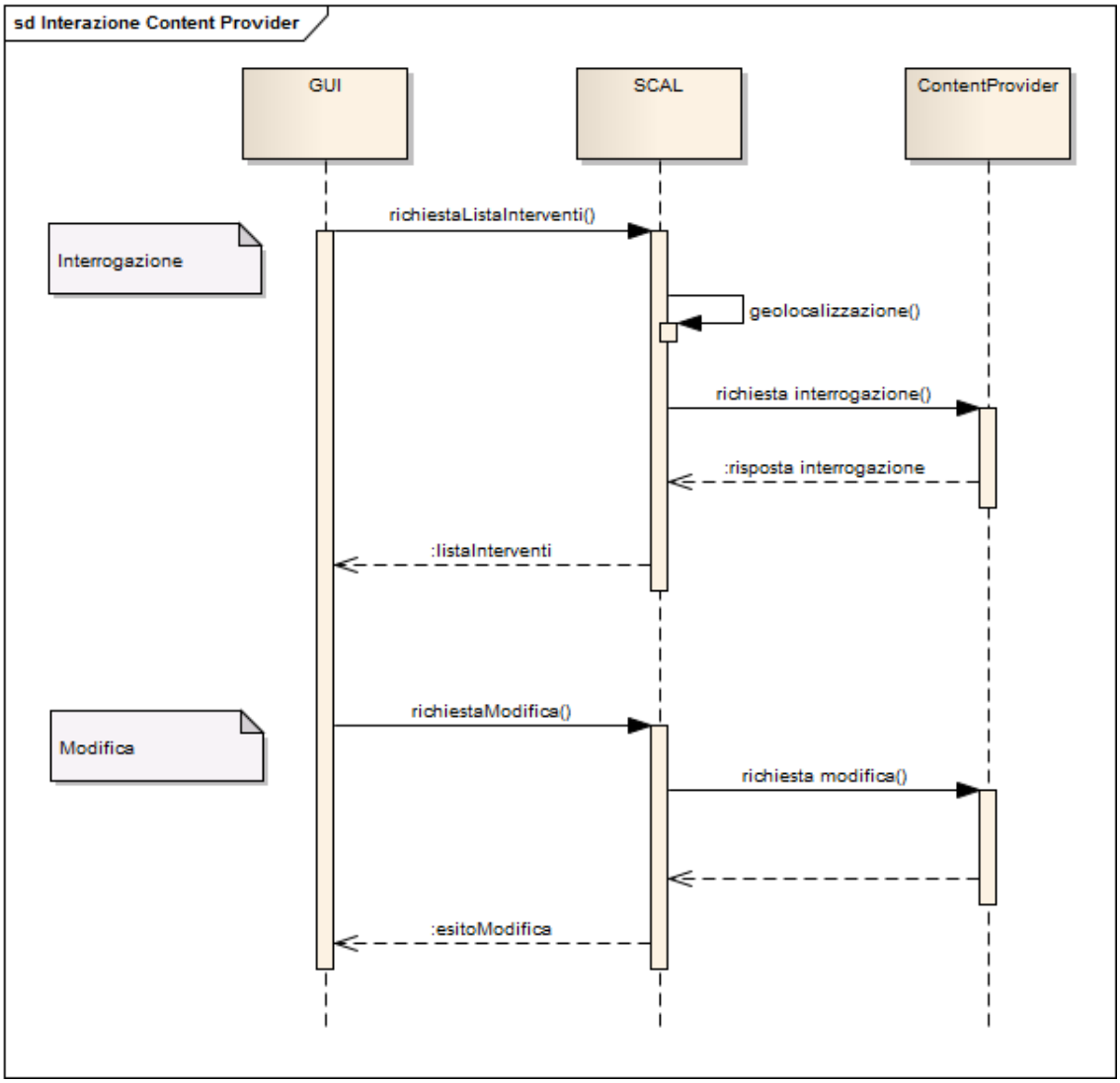

Figura 2.4: Struttura dell'interazione SmartClient-Content Provider

Il modo in cui gli oggetti del sistema collaborano tra loro determina cosa fa un sistema e, conseguentemente, la correttezza dell'esecuzione del sistema. La corretta collaborazione –o *interazione*– tra oggetti in un sistema è un punto critico della correttezza del sistema.

2.2 Scenari di interazione

L'interazione è una richiesta da parte di un soggetto(mittente) ad un altro (ricevitore) di eseguire una delle operazioni del ricevitore e tutto il processo svolto dal ricevitore per completare la richiesta [McGregor, Sykes 6]. Il diagrammi di sequenza aprono due diversi scenari di interazione, che presentano problematiche distinte.

2.2.1 Scenario 1: interazione con il Web Server

L'interazione tra l'Application Logic e il Web Server può essere definita in due casi:

- Il Web Server è già in opera per cui bisogna attenersi alle sue regole di interazione. In questo caso non verrà definito nessun nuovo linguaggio di interazione.
- C'è la possibilità di poter estendere/modificare il Web Server in modo da poter interagire tramite il linguaggio definito in questa sede.

Il secondo caso è ritenuto migliore in quanto rende l'interazione indipendente dal tipo di Web Server con cui si comunica; infatti nel primo caso, se per qualche motivo il Web Server venisse modificato, anche l'Application Logic dovrebbe essere modificata.

Fissata la necessità di specificare un linguaggio di comunicazione, bisogna definire chi risponde alle richieste dell'Application Logic. Anche qui si presentano due ipotesi:

- È il Web Server stesso che comunica con l'Application Logic tramite protocollo HTTP.
- È un'applicazione ad hoc (Adapter) che si occupa della comunicazione.

Tutte e due le ipotesi sono ritenute appropriate, ma in questo caso si decide di ricorrere all'Adapter, in modo da rendere la comunicazione completamente indipendente dall'interlocutore remoto. Teoricamente, con questa tecnica, al di là dell'Adapter potrebbe non trovarsi un Web Server, ma il sistema continuerebbe a funzionare ugualmente.

Il linguaggio da definire costituisce le specifiche del fornitore del servizio(in questo caso l'Adapter), l'IDL (Interface Definition Language). Dato che riguarda solo le specifiche, l'IDL è più semplice di un linguaggio di programmazione. Le specifiche dell'IDL forniscono molte informazioni che sono utili per scopi di collaudo [McGregor, Sykes 8]. L'interazione è a scambio di messaggi di tipo request-response. Per quanto riguarda l'interrogazione il modello è il seguente:

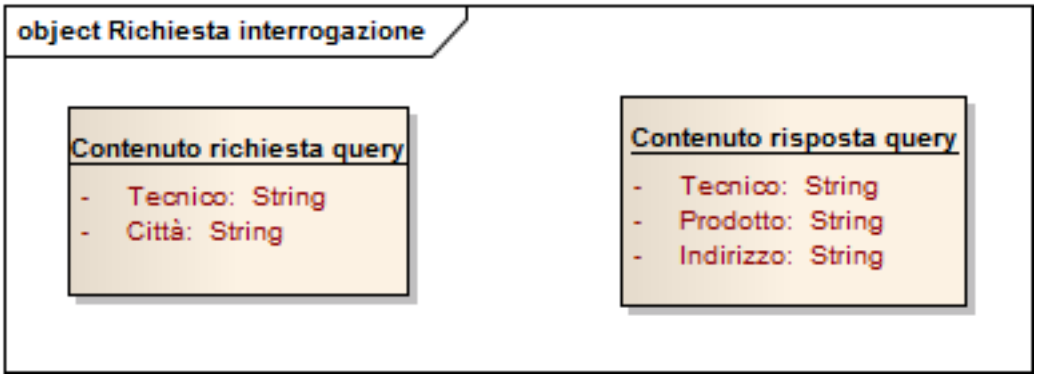

Figura 2.5: Modello interrogazione

- Contenuto della richiesta di query: l'AL invia all'Adapter il nome del tecnico e il nome della città in cui si trova.
- Contenuto della risposta di query: l'Adapter risponde con il nome del tecnico, il nome del prodotto su cui effettuare l'intervento e l'indirizzo.

In questo caso si ha bisogno di serializzare il modello, in modo da ottenerne una rappresentazione lineare. Si definisce quindi la sintassi della rappresentazione:
Contenuto della richiesta di query: "nome tecnico | nome città"
Contenuto della risposta di query: "nome tecnico | nome prodotto | indirizzo"
Una rappresentazione più formale verrà proposta nel capitolo successivo in cui si definiranno i piani di collaudo.

Il modello della richiesta di modifica dell'AL e della risposta dell'Adapter è il seguente:

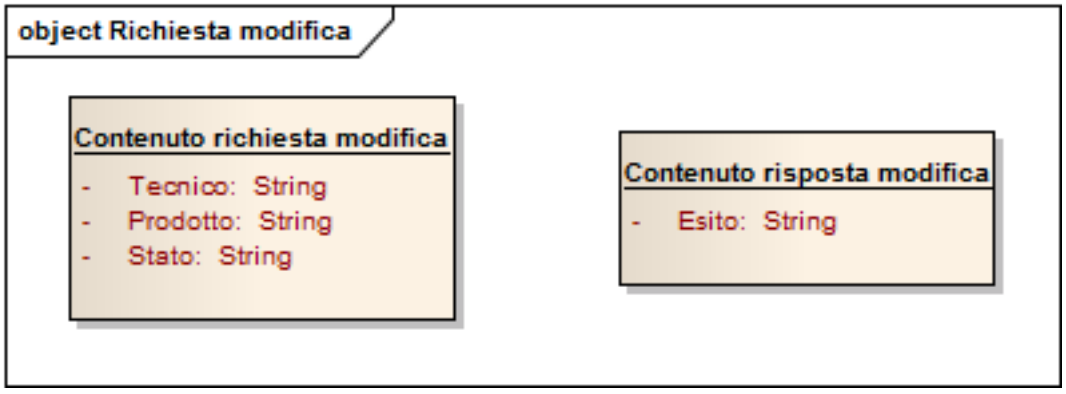

Figura 2.6: Modello modifica

- Contenuto della richiesta di modifca: l'AL invia all'Adapter il nome del tecnico, il nome del prodotto e il nuovo stato dell'intervento.
- Contenuto della risposta di modifica: l'Adapter risponde con l'esito dell'operazione, ovvero se è andata a buon fine oppure no.

Anche per questo modello si definisce una sintassi lineare di rappresentazione:
Contenuto della richiesta di modifica: "nome tecnico | nome prodotto | stato intervento"
Contenuto della risposta di modifica: "Modifica effettuata" (nel caso di esito positivo) oppure "Modifica non riuscita" (nel caso di esito negativo).

2.2.2 Scenario 2: interazione con il Content Provider

Il tipo di interazione in questo caso consiste nel metodo di una classe che si riferisce ad un'instanza globale di un'altra classe. Se una classe referenzia qualche oggetto globale, bisogna trattarlo come un paramtero implicito dei metodi che lo referenziano [McGregor, Sykes 6]. Una volta decisa l'esistenza di un Content Provider, è necessario pensare ad un modello di rappresentazione dei dati opportuno (facendo riferimento al modello del dominio descritto precedentemente).

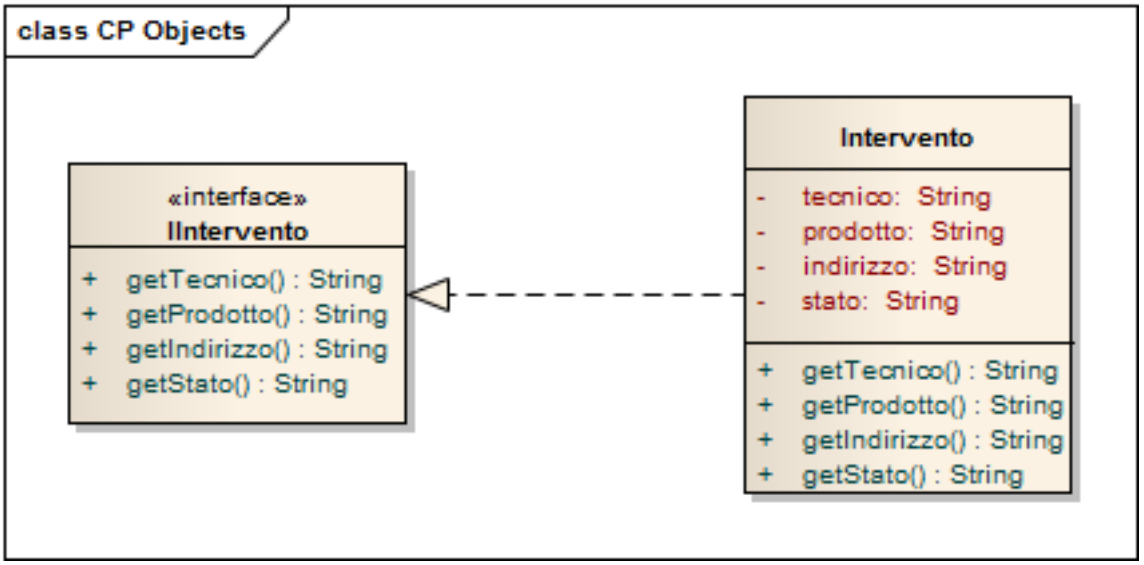

Figura 2.7: Modello dati

Questo modello dei dati è stato pensato per essere il più semplice possibile e funzionale al sistema. Il passo successivo consiste nel ricavare la tabella che andrà a popolare il Content Provider.

Figura 2.8: Tabella Intervento

La tabella è composta dagli stessi elementi di cui è composto il modello.

- idIntervento: Chiave primaria di tipo intero. Identificativo dell'intervento.
- tecnico: Stringa di lunghezza massima di 20 caratteri, non nulla. Nome del tecnico a cui è stato assegnato l'intervento.
- prodotto: Stringa di lunghezza massima di 20 caratteri, non nulla. Nome del prodotto su cui effettuare l'intervento.
- indirizzo: Stringa di lunghezza massima di 30 caratteri, non nulla. Indirizzo in cui si trova il prodotto.
- stato: Stringa di lunghezza massima di 10 caratteri. Stato di esecuzione dell'intervento.

Definito il modello di interazione del sistema, si passa alla stesura dei piani di collaudo.

Capitolo 3

Piano di collaudo

3.1 Tipi di collaudo

Durante il processo di sviluppo di un sistema, ci si avvale di piani di collaudo per definire e testare il sistema. In base alla fase del processo, si impiegano diversi tipi di collaudo, che hanno scopi ben distinti.

- Unit Test. L'obiettivo principale dello Unit Test è prendere la parte più piccola di software collaudabile nell'applicazione, isolarla dal resto del codice e determinare se si comporta esattamente come ci si aspetta. Ogni unità è collaudata separatamente prima di integrarla con altre in moduli. Lo Unit Test ha dimostrato il suo valore in quanto una larga percentuale di difetti vengono identificati durante il suo utilizzo.

- Integration Test. È un'estensione logica dello Unit Test. Nella sua forma più semplice, due unità che sono già state collaudate vengono combinate in un componente e l'interfaccia tra esse viene testata. Un componente, in questo senso, riferisce ad un aggregato di più unità. In uno scenario realistico, più unità vengono combinate in componenti, che a loro volta vengono integrati in parti più grandi del sistema. L'idea è di testare combinazioni di parti ed eventualmente estendere il sistema per testare i moduli con quelle di altri gruppi. Alla fine tutti i moduli che compongono un sistema vengono testati insieme.

Lo Unit Test è già stato realizzato durante la fase di analisi dei requisiti, mentre l'Integration Test viene presentato qui di seguito.
In ragione alle considerazioni fatte durante l'analisi del problema, il piano di collaudo si articola in questo modo:

- Piano di collaudo Application Logic-Web Server
- Piano di collaudo Application Logic-Content Provider

3.2 Application Logic-Web Server

L'interazione tra Application Logic e Web Server è costituita da uno scambio di messaggi tramite protocollo Http. Il piano di collaudo riguarderà quindi il meccanismo request-response tipico del protocollo Http.
Per prima cosa si definisce il setUp, in cui in fase di collaudo verranno inizializzati i componenti coinvolti.

```
public class WebServerActivityTest extends ActivityInstrumentationTestCase2<Activity> {

    //componente che si occupa di inviare le richieste Http e di ricevere le risposte
    private IHttpTask h1;

    public WebServerActivityTest() {
        super(Activity.class);
    }

    public void setUp() throws Exception{
        super.setUp();

        h1=null;
    }
```

Figura 3.1: setUp piano di collaudo Web Server

IHttpTask è l'elemento che si occupa di inviare le richieste Http all'Adapter e di ricevere le risposte.

```
public interface IHttpTask extends AsyncTask<String, Void, HttpResponse>{

    public HttpResponse doInBackground(String... params);
    public void onPostExecute(HttpResponse response);
    public String getResult();


}
```

Figura 3.2: IHttpTask

```
public interface AsyncTask<T1, T2, T3> {

    public AsyncTask<T1, T2, T3> execute(String... params);
}
```

Figura 3.3: AsyncTask

3.2.1 Interrogazione Web Server

La prima funzionalità da definire è quella di interrogazione dell'Adapter

```
public void testInterroga(){
    //1° caso: il contenuto della richiesta riferisce ad un dato presente
    //si invia la richiesta Http all'indirizzo in cui si trova l'Adapter
    h1.execute("indirizzo Adapter", "paolo|Cesena");
    //la risposta dell'Adapter deve essere uguale al contenuto del dato aspettato
    assertEquals(h1.getResult(),"paolo|computer|via Roma 3");

    //2° caso: il contenuto della richiesta non corrisponde a nessun dato
    h1.execute("indirizzo Adapter", "giorgio|Londra");
    //l'Adapter deve comunicare l'assenza di dati corrispondenti
    assertEquals(h1.getResult(),"Nessuna corrispondenza trovata");

}
```

Figura 3.4: Interrogazione Adapter

3.2.2 Modifica Web Server

La seconda funzionalità da definire è quella di modifica dell'Adapter

```
public void testModifica(){
    //si invia la richiesta Http all'indirizzo in cui si trova l'Adapter,
    //specificando il contenuto della richiesta di modifica
    h1.execute("indirizzo Adapter", "paolo|computer|eseguito");
    //l'Adapter deve rispondere con l'esito della modifica
    assertTrue(h1.getResult().equals("Modifica effettuata") |
               h1.getResult().equals("Modifica non riuscita"));
}
```

Figura 3.5: Modifica Adapter

Si può osservare che in questo piano di collaudo si mettono esplicitamente in evidenza i supporti per le interazioni, e quindi si anticipano dettagli implementativi. La stessa cosa accadrà per il piano di collaudo del Content Provider.

3.3 Application Logic-Content Provider

Definizione del setUp del piano.

```
public class ContentProviderActivityTest extends ProviderTestCase2<ContentProvider> {

    private ContentProvider cp;
    private Cursor cursor;
    private ContentValues values;
    //URI per accedere al Content Provider
    private final static Uri LIST_INTERVENTO_URI=
            Uri.parse("content://it.tesi.interventocontentprovider/intervento#");


    public ContentProviderActivityTest() {
        super(ContentProvider.class,"it.tesi.interventocontentprovider");
    }

    protected void setUp() throws Exception{
       super.setUp();
       //in questo punto si eseguirà il popolamento iniziale del CP
       cp=null;
    }
```

Figura 3.6: setUp piano di collaudo Content Provider

3.3.1 Creazione Content Provider

Definizione della creazione del Content Provider.

```
    public void testCreazione(){
      //se il CP è stato creato,
      //l'operazione di query al CP deve tornare un Cursore non vuoto
      cursor=cp.query(LIST_INTERVENTO_URI,null,null,null,null);
      assertNotNull(cursor);
      cursor.close();
    }
```

Figura 3.7: creazione Content Provider

3.3.2 Interrogazione Content Provider

Prima funzionalità: interrogazione del Content Provider.

```
public void testInterroga(){
   //1° caso: l'interrogazione va a buon fine, esiste una tupla corrispondente
   String [] selectionArgs={"giulio"};
   //si inserisce nel CP una nuova tupla
   values.put("tecnico", "giulio");
   values.put("prodotto", "p1");
   values.put("stato", "eseguito");
   values.put("indirizzo", "via foscolo 20");
   cp.insert(LIST_INTERVENTO_URI, values);
   //si esegue la query specificando il tecnico
   //che era stato inserito precedentemente
   cursor=cp.query(LIST_INTERVENTO_URI,null,"tecnico=?",selectionArgs,null);
   cursor.moveToFirst();
   //la query deve tornare la tupla inserita,
   //getString(1) si riferisce al campo "prodotto"
   assertEquals(cursor.getString(1),"p1");
   cursor.close();

   //2°caso: l'interrogazione non va a buon fine
   String [] selectionArgs2={"alessio"};
   //si esegue la query specificando un tecnico non presente
   cursor=cp.query(LIST_INTERVENTO_URI,null,"tecnico=?",selectionArgs2,null);
   //la query deve tornare null, perchè non ha trovato nessuna corrispondenza
   assertNull(cursor);
}
```

Figura 3.8: interrogazione Content Provider

3.3.3 Modifica Content Provider

Seconda funzionalità: modifica del Content Provider.

```
public void testModifica(){
   String [] selectionArgs={"luca"};
   values.put("stato", "eseguito");
   //si modifica la tupla cambiando il campo "stato"
   cp.update(LIST_INTERVENTO_URI, values, "tecnico=?", selectionArgs);
   //si esegue la query sulla tupla modificata
   cursor=cp.query(LIST_INTERVENTO_URI,null,"tecnico=? ",selectionArgs,null);
   cursor.moveToFirst();
   //getString(0) si riferisce al campo "stato"
   assertEquals(cursor.getString(0),"eseguito");
   cursor.close();
}
```

Figura 3.9: modifica Content Provider

3.4 Osservazioni

A seguito dell'analisi degli scenari di interazione del sistema, è stato ritenuto opportuno realizzare un piano di collaudo per ogni funzionalità che il sistema implementerà. Osservando il risultato emerge che il piano di collaudo relativo al Web Server assume una forma molto diversa da quella del piano di collaudo relativo al Content Provider, nonostante la funzionalità da definire sia la stessa. Si nota infatti che per ogni funzionalità stabilita (interrogazione e modifica) sono stati realizzati due differenti piani di collaudo, per due differenti contesti operativi (Web Server e Content Provider), e quindi si è mescolata la funzionalità con la tecnologia di interazione.
Da un punto di vista ingegneristico, non è accettabile dettagliare in fase di pianificazione del collaudo come dovrà lavorare il sistema, ovvero come realizzerà le funzioni che deve fornire. Un corretto piano di collaudo dovrebbe definre *cosa* deve essere realizzato dal sistema in relazione alle funzionalità individuate senza ancora specificare *come* lo farà.

In luce a queste considerazioni, si tenta di fornire un modello di architettura logica che si ponga ad un livello più alto di astrazione, per poi definire un piano di collaudo in linea con i principi esposti.

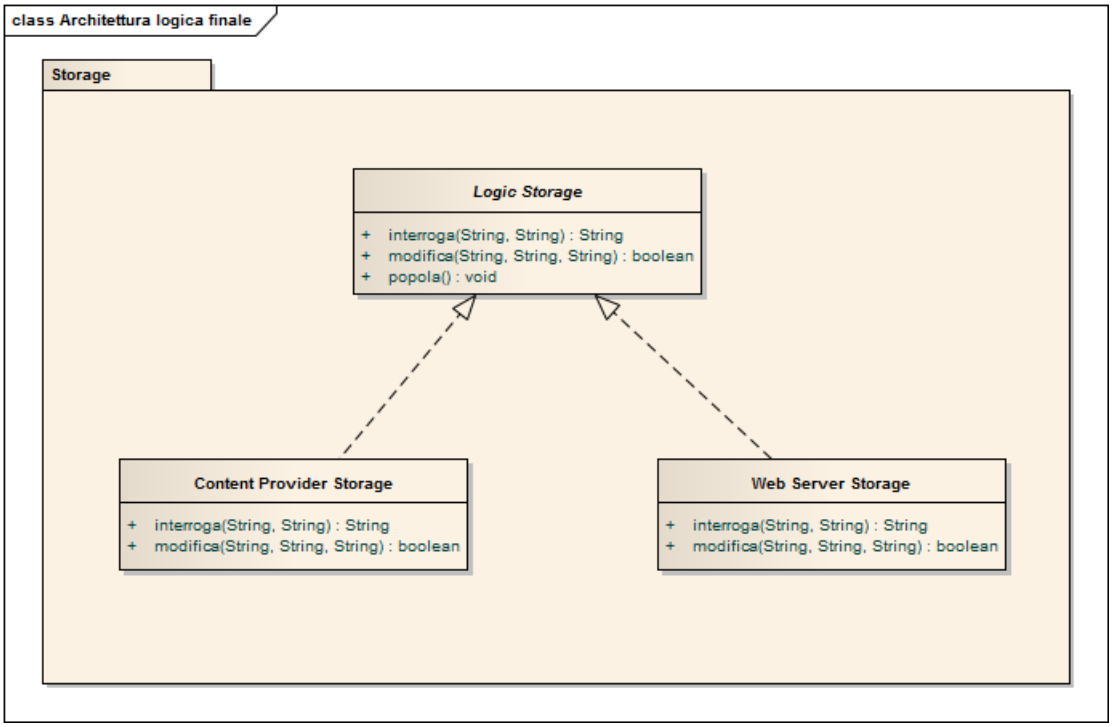

Figura 3.10: Architettura logica Storage

Il Logic Storage è una classe astratta che racchiude le funzionalità che deve supportare lo storage di dati. Content Provider Storage e WebServer Storage sono viste come possibili implementazioni di tale astrazione. Questa struttura permette di inserire in ogni momento una nuova implementazione che soddisfi le particolari esigenze.
Per completare la descrizione delle funzionalità viene proposto un piano di collaudo, che definisce univocamente che cosa dovrà fare lo storage di dati, senza anticipare come verrà fatto.

```
public abstract class LogicStorage {

    public abstract void popola();
    public abstract String interroga(String tecnico, String citta);
    public abstract boolean modifica(String tecnico, String prodotto, String stato);

}
```

Figura 3.11: Logic Storage

Si assume che lo storage sia popolato da almeno questo record: “luca|computer|via Roma 4 Cesena|da eseguire”

```
public class LogicStorageActivityTest extends ActivityInstrumentationTestCase2<Activity> {

    private LogicStorage store;

    public LogicStorageActivityTest() {
        super(Activity.class);
    }

    public void setUp() throws Exception{
        super.setUp();

        store=null;
        store.popola();
    }
```

Figura 3.12: SetUp LogicStorage

```
    public void interroga(){

        //1° caso: i dati inseriti trovano corrispondenza nello storage
        String answer=store.interroga("luca", "Cesena");
        assertEquals(answer, "luca|computer|via Roma 4");

        //2° caso: i dati inseriti non trovano corrispondenza
        String answer2=store.interroga("paolo", "Londra");
        assertEquals(answer2,"Nessuna corrispondenza trovata");
    }

    public void testModifica(){

        //1° caso: la modifica va a buon fine
        //perchè i dati inseriti trovano corrispondenza
        boolean answer=store.modifica("luca", "computer", "eseguito");
        assertTrue(answer);

        //2° caso: la modifica non va a buon fine
        //perchè i dati inseriti non trovano corrispondenza
        boolean answer2=store.modifica("paolo", "fax", "eseguito");
        assertTrue(!answer2);
    }
```

Figura 3.13: Funzionalità LogicStorage

Guardando il piano di collaudo appena formulato, ci si accorge che i piani di collaudo realizzati precedentemente non sono altro che una specializzazione di questo. Con la stessa logica è quindi possibile creare nuove implementazioni del Logic Storage e collaudarle tramite piani di collaudo specializzati.

Capitolo 4

Progetto e implementazione

4.1 Progetto

Dopo la formalizzazione dei piani di collaudo, è immediato definire i componenti che andranno a costituire il sistema:

- GUI e Application Logic. La GUI è costituita dall'interfaccia grafica con cui l'utente potrà accedere alle funzionalità del sistema. Prevede la selezione della modalià con cui si effettua la richiesta (locale/remoto) e la selezione del tipo di richiesta da effettuare (interrogazione/modifica). Una volta selezionato il tipo di richiesta apparirà la schermata corrispondente con i campi opportuni da compilare.
  L'Application Logic avrà il compito di gestire i cambi di schermata, accedere alle funzionalità ed eseguire le richieste all'Adapter per la modalità in remoto e al Content Provider per la modalità in locale.

- Content Provider. Rappresenta lo storage di informazioni in locale. Realizza le funzionalità di accesso e modifica dei dati.

- Adapter (Mock Object). È il componente che fa da ponte tra l'Application Logic e il Web Server, rendendo indipendente l'interazione. In questo caso di studio il Web Server non verrà implementato, ma verrà sostituito da un Mock Object incapsulato nell'Adapter. Il Mock Object si sostituisce a un componente reale implementandone la stessa interfaccia. Esso permette di testare le interazioni con il componente chiamante in quanto incapsula quello che è lo stato che ci si aspetta a seguito di una particolare interazione, utilizzandolo per far fallire o meno un test [Carli].

L'interazione tra i componenti del sistema è visualizzata nei diagrammi di sequenza.

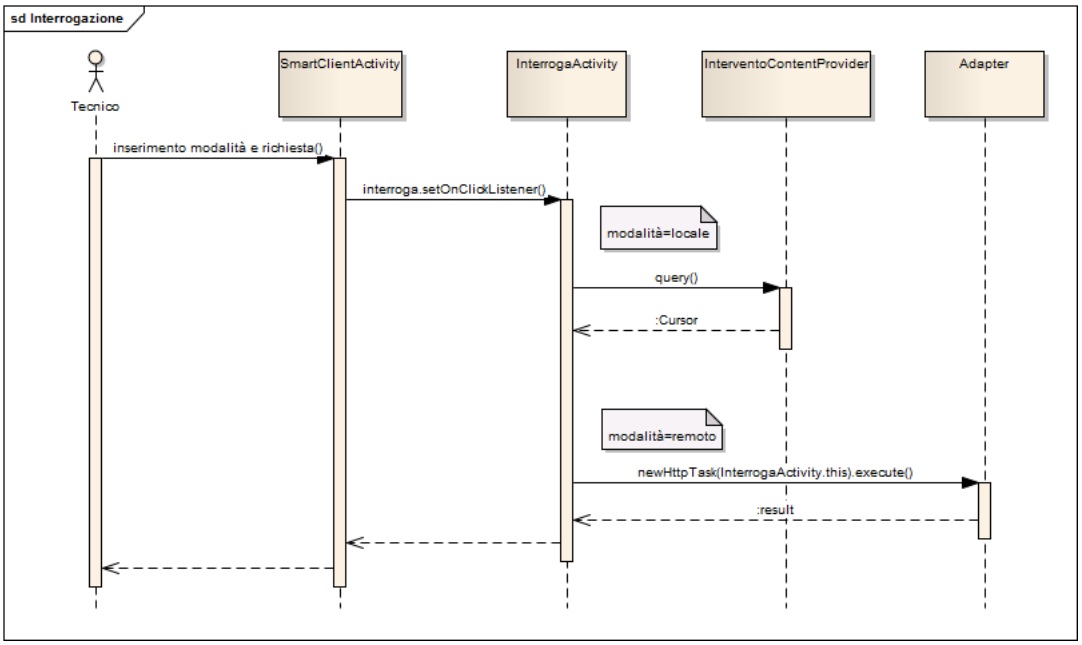

Figura 4.1: Interrogazione

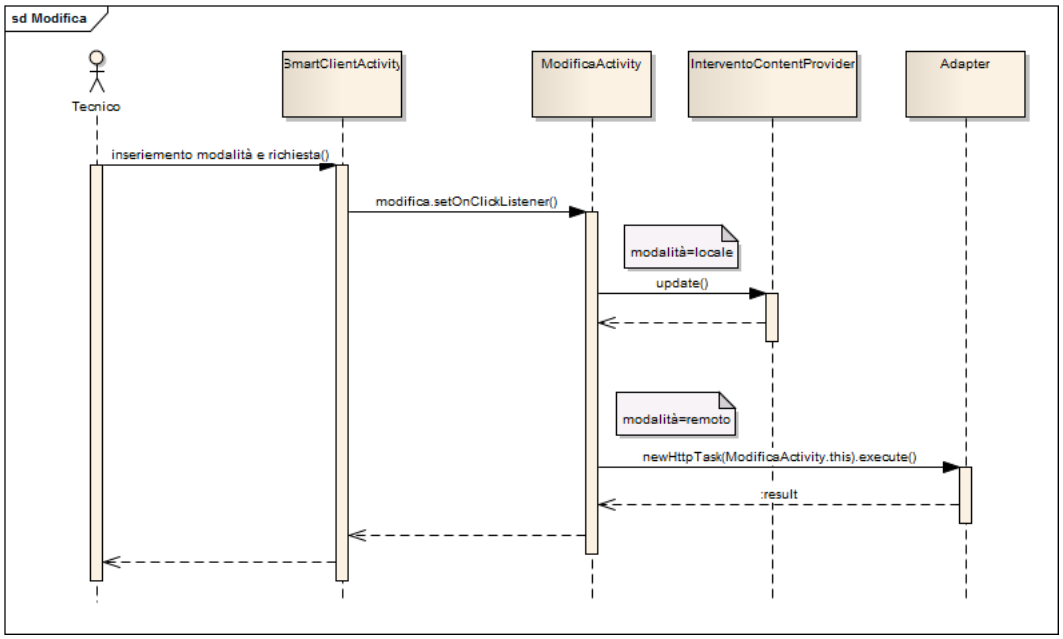

Figura 4.2: Modifica

4.2 Implementazione

L'ultima fase del processo di sviluppo consiste nell'implementazione del sistema, in cui i risultati della progettazione vengono messi in pratica. Grazie alla formalizzazione di tutti gli aspetti del sistema, che ha caratterizzato ogni fase dello sviluppo sarebbe possibile delegare l'implementazione ad un team di persone, ognuno dei quali si occuperebbe della realizzazione di un modulo del sistema. Per ovvi motivi in questo caso il sistema è stato implementato da una sola persona, ma sarebbe stato possibile optare per uno sviluppo in team.
L'applicazione è stata realizzata completamente in ambiente Android come specificato nei requisiti, tranne l'Adapter, implementato tramite una Servlet. Come definito in fase di progettazione, l'Adapter incapsula un Mock Object, quindi la Servlet risponderà alle richieste dell'Application Logic con stringhe predefinite, sia per la richiesta di interrogazione, sia per la richiesta di modifica.
Una volta terminata l'implementazione, l'applicazione è stata testata su un device fisico.

Conclusioni

In questo elaborato è stato affrontato il tipico percorso del processo di produzione di un sistema software, partendo dall'analisi dei requisiti, passando per l'analisi del problema e il piano di collaudo, per finire con la progettazione e l'implementazione del sistema. Si è cercato di evidenziare come questo processo non sia lineare, di come a seguito di nuove considerazioni la struttura del sistema sia cambiata, anche contraddicendo osservazioni precedenti.
Un grande beneficio di questo approccio alla progettazione è che i modelli dell'analisi mappano i modelli del progetto che, a loro volta, mappano il codice. Così, si può iniziare il piano di collaudo durante l'analisi e perfezionarlo in sede di progettazione. Il piano di collaudo della progettazione verrà perfezionato a sua volta per l'implementazione. Questo significa che un processo di collaudo può essere intrecciato con il processo di sviluppo. Possono essere individuati diversi vantaggi portati da questo metodo [McGregor,Sykes 1]:

- Collaudare precocemente aiuta gli analisti e i progettisti a capire e esprimere meglio i requisiti e assicurarsi che questi specifici requisiti siano collaudabili.
- I difetti possono essere rilevati subito nel processo di sviluppo, risparmiando tempo, soldi e fatica.
- Modellare il collaudo assicura che i collaudatori e gli sviluppatori abbiano una consistente conoscienza dei requisiti del sistema.

Il punto chiave della tesi è l'importanza che ricopre il piano di collaudo all'interno del processo di produzione. Esso non è solo un modo per realizzare il collaudo del sistema, anzi si è sottolineato il fatto che il piano di collaudo può fornire una descrizione univoca di *cosa* deve fare il sistema. Con altri metodi non è possibile raggiungere il grado di formalità che si raggiunge con il piano di collaudo. Inoltre il piano di collaudo, insieme ai modelli del sistema, aiutano a capire la misura dei costi necessari per eseguire

adeguati test di sistema, determinando cosa necessita di essere testato.
Una prova a sostegno della tesi affermata si può trovare in numerosi libri che trattano della cosiddetta "crisi del software". La locuzione "crisi del software" venne ideata quando i progetti erano continuamente in ritardo e oltre il budget prestabilito. I progetti erano in ritardo perchè l'applicazione era complessa e scarsamente compresa sia dai clienti che dagli sviluppatori e nessuno aveva alcuna idea di come stimare la difficoltà del lavoro e di quanto tempo ci sarebbe voluto per risolverlo. La storia mostra la crescita dell'ingegneria del software partendo dalla programmazione. Alcune evoluzioni tecnologiche hanno giocato un ruolo importante nello sviluppo della disciplina. L'influenza maggiore è stata il cambio di equilibrio tra i costi dell'hardware e del software. Un'altro orientamento evolutivo si è creato anche all'interno del campo stesso. Vi è stata una crescente enfatizzazione a veder l'ingegneria del software come una disciplina che va ben oltre la pura attività di codifica. Al contrario, il software viene considerato un prodotto che ha un intero ciclo di vita, partendo dalla sua concezione, continuando attraverso la progettazione, lo sviluppo, la messa in funzione, la manutenzione e l'evoluzione. Lo spostamento dell'enfasi dalla codifica all'intero ciclo di vita del software ha favorito lo sviluppo di metodologie e di sofisticati strumenti a sostegno delle squadre coinvolte [Ghezzi, Jazayeri, Mandrioli].
Il piano di collaudo è quindi un indispensabile tassello di questo ciclo di vita.

Bibliografia

[3-tier] *Architettura three-tier*. URL: http://it.wikipedia.org/wiki/Architettura_three-tier/

[McGregor,Sykes 1] John D. McGregor, David A. Sykes: "Chapter 1:Introduction". *A Practical Guide to Testing Object-Oriented Software*, Addison-Wesley Professional.

[McGregor, Sykes 6] John D. McGregor, David A. Sykes: "Chapter 6:Testing Interactions". *A Practical Guide to Testing Object-Oriented Software*, Addison-Wesley Professional.

[McGregor, Sykes 8] John D. McGregor, David A. Sykes: "Chapter 8:Testing Distributed Objects". *A Practical Guide to Testing Object-Oriented Software*, Addison-Wesley Professional.

[Carli] M. Carli: "Capitolo 4.Test e Instrumentation". *Sviluppare applicazioni per Android*, APOGEO.

[Ghezzi, Jazayeri, Mandrioli] Ghezzi, Jazayeri, Mandrioli: "Capitolo 1.Ingegneria del software: visione d'insieme". *Ingegneria del software: fondamenti e principi*, PEARSON.